IMPLEMENTASI *LOAD BALANCING* SERVER BASIS DATA PADA VIRTUALISASI BERBASIS KONTAINER

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Moch Wahyu Imam Santosa
NIM: 145150201111170

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI *LOAD BALANCING* SERVER BASIS DATA PADA VIRTUALISASI BERBASIS KONTAINER

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Moch Wahyu Imam Santosa
NIM: 145150201111170

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Rakhmadhany Primananda, S.T., M.Kom,	Widhi Yahya, S.Kom., M.T.,
NIK. 2016098604061001	NIK. 2016078911211001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan,. S.T, M.T. Ph.D.,
NIP: 197105182003121001

ii

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 19 Juli 2018

METERAI TEMPEL
48E40AFF198905692
6000
ENAM RIBU RUPIAH

Moch Wahyu Imam Santosa
NIM: 145150201111170

iii

KATA PENGANTAR

Puji syukur Alhamdulillah penulis panjatkan kehadirat Allah yang telah memberikan rahmat dan hidayah-Nya serta memudahkan dalam penulisan laporan ini hingga dapat menyelesaikan skripsi ini dengan tepat pada waktunya. Shalawat serta salam juga senantiasa tetap terlantunkan pada guru besar, Kyai dan juga panutan umat manusia yakni baginda nabi Muhammad Salallahualaihi wasalam.

Laporan skripsi dengan judul "Implementasi *Load Balancing* Basis Data Server Pada Virtualisasi Berbasis *Container*" tidak dapat terselesaikan tanpa bantuan dari berbagai pihak. Dalam kesempatan ini pula penulis mengucapkan terima kasih dan penghargaan sebesar-besarnya atas bantuan, motivasi dan bimbingan yang diberikan, antara lain kepada :

1. Bapak Rakhmadhany Primananda S.Kom., M.Kom., dan Bapak Widhi Yahya S.Kom., M.Sc selaku dosen pembimbing skripsi yang telah membimbing dan mengarahkan penulis untuk dapat menyelesaikan skripsi ini dengan baik.
2. Seluruh jajaran Pimpinan, Dosen, serta seluruh civitas Fakultas Ilmu Komputer yang membantu serta mendukung pembuatan skripsi ini.
3. Kedua orang tua. Bapak Slamet Riyadi dan juga Ibu Sumarmi serta kedua adik, Dimas Wahyu Adi Budiman dan Anggieta Triwahyu Firtia Permatasari serta Dyah Ayu Ocky Mawardani yang selalu memberikan support, dukungan moral dan material kepada penulis untuk dapat terselesainya skripsi ini.
4. Rekan-rekan MCU, Bapak ibnal Affan, Bapak Juarij Migdad atas semangat dan support dalam membantu dan memberikan saran untuk pengerjaan skripsi ini.
5. Dinas Komunikasi dan Informatika kabupaten Sampang melalui Bapak Amrin Hidayat selaku Kabid TIK yang berkenan meminjamkan seluruh infrastruktur yang dimiliki untuk membantu penelitian didalam skripsi ini.
6. Riski Pradana, Robihamanto, Mukmin, Eko Setiyono, dan Bosarito yang selalu memberikan masukan dan saran.
7. Semua yang tidak dapat disebutkan satu persatu.

Semoga segala bantuan yang telah diberikan kepada penulis mendapat pahala dari Allah. Penulis menyadari bahwa skripsi ini tidaklah sempurna karena keterbatasan materi dan pengetahuan yang dimiliki penulis, sehingga penulis menerima saran dan kritikan yang dapat disampaikan melalui *e-mail*. Akhirnya, besar harapan bahwa skripsi yang telah ditulis dapat bermanfaat bagi pembaca.

Malang, 19 Juli 2018

Penulis
me@imamsantosa.id

ABSTRAK

Moch Wahyu Imam Santosa, Implementasi *Load Balancing* Server Basis Data Pada Virtualisasi Berbasis Kontainer.

Pembimbing : Rakhmadhany Primananda, S.T., M.Kom., dan Widhi Yahya, S.Kom., M.Sc

Sistem yang saat ini banyak digunakan masih banyak menerapkan arsitektur *single* server basis data. Hal tersebut menyebabkan ketidakmampuan server basis data dalam menangani permintaan data yang besar. *Horizontal scalling* dengan melakukan penambahan unit pemrosesan seperti *node, instance* ataupun kontainer merupakan sebuah solusi dalam meningkatkan kinerja sebuah sistem dalam menangani banyaknya permintaan. *Load balancer* dibutuhkan sebagai sistem yang digunakan untuk melakukan distribusi permintaan data kepada sistem server basis data. Dalam penelitian ini penulis mencoba untuk menyelesaikan permasalahan tersebut dengan melakukan implementasi *load balancing* server basis data pada virtualisasi berbasis kontainer. Server basis data PostgreSQL diisolasi didalam sebuah kontainer berserta komponennya. Kemudian dilakukan *deployment* kontainer basis data tersebut kedalam *cluster* yang dikelola oleh Kubernetes. Kubernetes menangani segala aktifitas yang dibutuhkan oleh *service* dari kontainer PostgreSQL seperti *load balancing. Load balancing* server basis data ini menggunakan fitur *load balancer dan nodeport* yang dimiliki oleh Kubernetes. Penelitian ini juga melakukan percobaan replikasi kontainer server basis data sebanyak maksimal 5 replika yang kemudian dari setiap replika terkoneksi pada sebuah sistem penyimpanan menggunakan *Network File System (NFS)*. Dengan terhubungnya setiap replika pada sebuah sistem penyimpanan terpusat menjadikan data yang dikirimkan memiliki kesamaan atau konsisten. Dari hasil pengujian yang telah dilakukan menunjukan bahwa dengan menambah unit pemrosesan dapat meningkatan kinerja server basis data PostgreSQL dalam melayani kebutuhan data yang diminta oleh pengguna. Hasil *troughput* pada pengujian *load balancing* dengan beban 750 *request* per detik didapatkan hasil (61,3 *request* per detik) pada 2 replika, (63,1 *request* per detik) pada 3 replika, (64,6 *request* per detik) pada 4 replika, dan (65,6 *request* perdetik) pada 5 replika. Selain itu didapatkan hasil distribusi data secara merata pada setiap replika yang tersedia.

Kata kunci: *load balancer*, kontainer, replikasi, basis data PostgreSQL

ABSTRACT

Moch Wahyu Imam Santosa, Implementasi *Load Balancing* Server Basis Data Pada Virtualisasi Berbasis Kontainer.

Pembimbing : Rakhmadhany Primananda, S.T., M.Kom., dan Widhi Yahya, S.Kom., M.Sc

Most of systems that we use nowadays still implement single database server architecture. It makes database server incapable in handling many requests. Horizontal scaling that is done with additional processing units such as nodes, instances, or even containers is a solution in increasing performance of a system in handling many requests. Load balancer is needed as a system that will be used to distribute data requests to the system of database server. In this research, the author tried to solve the problem by implementing load balancing database server in container-based virtualization. Database server PostgreSQL was isolated in a container along with its components. And then, database server was deployed into cluster handled by Kubernetes. Kubernetes handles all of the activities needed by PostgreSQL container's service such as load balancing. Load balancing of this database server using load balancer and nodeport features that provided by Kubernetes. In this research, the database container was replicated up to 5 replicas which then each of them connected to a centralized storage system using Network File System (NFS). With each of the replicas connected in a centralized storage system, the data that was sent has similarity or consistency. From the research, it can be concluded that by adding processing units can increase the performance of PostgreSQL database server in handling data requested by users. The result shows that the performance of database server is increased, with throughput that was obtained from load of 750 per second was (61,3 request per second) in 2 replicas, (61,3 request per second) in 3 replicas, (64,6 request per second) in 4 replicas, and (65.,6 request per second) in 5 replicas.

Keywords : load balancer, container, replication, PostgreSQL database

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Di era globalisasi saat ini, kebutuhan data semakin meningkat. Berdasarkan data dari *Extrapolation of facebook data* bahwa pada bulan April tahun 2017 pengguna facebook dari Indonesia mencapai angka 111 juta pengguna. Angka tersebut menempati posisi ke empat terbesar dari peringkat dunia. Asosiasi penyelenggaraan jasa internet Indonesia (eMarketer, 2018) juga menyatakan pada tahun 2017 pengguna internet di Indonesia mencapai angka 54,68 persen dari total 262 juga populasi penduduk di Indonesia. Hal ini dapat dilihat bahwa saat ini masyarakat sangat bergantung pada data. Sebagian besar dari arsitetur yang digunakan saat ini hanya berfokus kepada meningkatkan kinerja web server dengan menggunakan *single backend* server basis data (Affan, 2018). Permasalahan yang kemudian muncul adalah bagaimana *single* server basis data tersebut mampu menangani lonjakan permintaan data yang sangat banyak. Seperti dalam kasus pengumuman penerimaan mahasiswa baru yang setiap tahun diadakan. Terjadi waktu tunggu yang cukup lama dikarenakan ketidak mampuan *server* basis data dalam menghadapi lonjakan data yang sangat besar (Prasetyo, 2018).

Dalam penelitian yang berjudul *Performing real-time social recommendations on a highly-available graph database cluster* terdapat dua solusi untuk menangani beban permintaan yang besar secara terus yaitu *Horizontal scalling* dan *vertical* (Constantinov, Potreas, & Mocanu, 2016). *Vertical scalling* saat ini sudah mulai ditinggalkan dikarenakan hal itu akan bergantung pada kemampuan dalam perkembangan aplikasi basis data tersebut. Sedangkan *horizontal scalling* menjadi topik hangat penelitian oleh berbagai pihak. *Horizontal Scalling* tidak bergantung kepada aplikasi database namun bergantung pada sebuah sistem yang saling terhubung. Dalam penelitian tersebut tidak dijelaskan bagaimana penanganan *Horizontal Scalling* tersebut.

Loadbalancing merupakan salah satu metode dari pengembangan yang dilakukan dengan *horizontal scalling*. *Load balancing* menjadi sebuah tantangan yang dapat dicoba melihat dari sisi peningkatan luar biasa dari sisi performa dalam melayani permintaan data, pemanfaatan sumber daya secara efisien (Vasudevan, 2015). Dengan menambahkan sistem pemprosesan baik berupa node, *instance* atau kontainer dianggap lebih mampu untuk meningkatkan kemampuan server basis data dalam memenuhi permintaan data yang dengan jumlah yang banyak secara terus menerus. Namun penambahan sistem pemprosesan tidak serta merta dapat menyelesaikan masalah. Muncul kembali masalah baru yaitu masalah mahalnya biaya yang harus dikeluarkan terhadap kebutuhan tersebut.

Berangkat dari berbagai permasalahan tersebut menjadikan dasar penulis untuk melakukan penelitian dengan mengimplementasi sebuah sistem *load balancing* server basis data PostgreSQL pada virtualisasi berbasis kontainer. Kontainer merupakan sebuah konsep untuk mengemas sebuah sistem yang terisolasi seperti layaknya virtual mesin dalam virtualisasi (Dirgantara, 2016). Kontainer memiliki keunggulan yaitu memiliki ukuran yang kecil serta dan dapat

dengan mudah untuk dilakukan pemindahan kedalam lingkungan sistem yang lain serta mendukung replikasi. Dengan memanfaatkan kemudahan yang disediakan oleh Kubernetes dalam melakukan *management* kontainer serta fitur *loadbalancing* dan konsep *distribute system.* Kubernetes merupakan sebuah sistem *open source* untuk mengoptimalisasi penyebaran, *scalling* dan pengelolaan kontainer yang kemudian dikelompokan dalam bentuk *logical unit* (kubernetes, 2018). Harapannya dengan penelitian ini dapat membantu menyelesaikan permasalahan.

1.2 Rumusan masalah

Berdasarkan Pemaparan latar belakang diatas, maka rumusan masalah yang bisa dikaji adalah sebagai berikut :

1. Bagaimana rancangan sistem *cluster* kontainer basis data PostgreSQL yang dikelola oleh Kubernetes?
2. Bagaimana rancangan sistem *load balancing* kontainer basis data PostgreSQL yang dikelola oleh Kubernetes?
3. Bagaimana kinerja sistem *load balancing* cluster kontainer basis data PostgreSQL yang dikelola oleh Kubernetes?

1.3 Tujuan

Dari rumusan masalah yang sudah dibuat, maka didapatkan tujuan dari penelitian ini dilaksanakan :

1. Dapat merancang sebuah sistem *cluster* kontainer basis data PostgreSQL yang dikelola oleh Kubernetes.
2. Dapat merancang sistem *load balancing* kontainer basis data PostgreSQL yang dikelola oleh Kubernetes.
3. Dapat mengetahui kinerja *loadbalancing* kontainer basis data PostgreSQL yang dikelola oleh Kubernetes.

1.4 Manfaat

Diharapkan dalam penelitian ini dapat membantu memecahkan masalah dalam meningkatkan kemampuan sistem basis data server dalam melayani permintaan data yang banyak. Sehingga diperoleh sebuah sistem *load balancing* basis data server berbasis kontainer yang mampu melayani permintaan data yang besar. Nantinya diharapkan mampu meningkatkan ketersediaan tingkat tinggi server basis data.

1.5 Batasan masalah

Agar pembahasan penelitian ini tidak melewati batas dari yang telah dirumuskan, maka diperlukan batasan-batasan berupa :

1. Implementasi sistem cluster kontainer basis data PostgreSQL
2. Kontainer dikelola oleh Kubernetes
3. Menggunakan fitur *loadbalancing* yang dimiliki oleh Kubernetes
4. Menggunakan *Network File System* sebagai penyimpanan data
5. Menggunakan data dummy untuk mengisi sistem basis data

6. Melakukan ujicoba *load balancing* dalam membaca data dari sistem basis data maksimal 1000 baris data
7. Melakukan replikasi maksimal 5 replika.

1.6 Sistematika pembahasan

Sistematika Penulisan laporan skripsi ini dijelaskan sebagai berikut :

1. BAB I Pendahuluan

   Pada bab ini membahas tentang latar belakang yang menjadikan landasan penelitian ini dibuat. Kemudian tentang rumusan masalah, tujuan dan Batasan masalah serta metodologi penelitian dan sistematika pembahasan.
2. BAB II Landasan Kepustakaan

   Berisi pembahasan tengang kajian pustaka dan dasar teori yang digunakan sebagai rujukan untuk melakukan penelitian dan merupakan inti dari teori yang digunakan dalam penelitian
3. BAB III Metodologi Penelitian

   Pada bab ini membahasan metode yang digunakan saat melakukan penelitian yang menguraikan tentang objek penelitian, variabel, Peranangan Sistem, Implementasi Sistem, Uji Coba Sistem, serta kesimpulan
4. BAB IV Rekayasa Kebutuhan dan Perancangan

   Bab IV mengenai rekasaya kebutuhan dan perancangan berisi tahapan yang dilakukan untuk menganalisis segala kebutuhan yang diperlukan dalam perancangan sistem dan implementasi.
5. BAB V Implementasi

   Berisi implementasi terkait dengan topik yang sedang diteliti. Sistem diimplementasi berdasarkan hasil perancangan yang telah dibuat pada bab IV.
6. BAB VI Pengujian dan Analisis

   Pada bab ini membahas proses dan melakukan analisa terhadap hasil pengujian.
7. BAB VII Penutup

   Pada bab ini berisi kesimpulan dan saran dari pembahasan penelitian ini yang diharapkan bermanfaat untuk penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini dijelaskan teori-teori pendukung penelitian yang sedang dilakukan seperti kontainer, PostgreSQL, Kubernetes, docker, *load balancing,* serta teori lainnya termasuk didalamnya kajian pustaka dari penelitian-penelitian yang telah dilakukan sebelumnya. Landasan kepustakaan dijadikan sebagai landasan teori-teori dalam mendukung penelitian serta mendukung dalam melakukan implementasi.

2.1 Kajian pustaka

Kajian pustaka menjelaskan terkait dengan kajian kepustakaan yang digunakan sebagai acuan serta perbandingan terhadap penelitian yang sedang dilakukan. Pada tabel 2.1 disampaikan bahwa penelitian yang sedang dilakukan merujuk dari pengembangan yang telah dilakukan.

Tabel 2.1 Kajian Pustaka dari penelitian sebelumnya

No.	Nama Penulis, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana Penelitian
1	Calin Constantinov, Cosmin M. Poteras, Mihai L. Mocanu. [2016]. *Performing real-time social recommendations on a highly-available graph database cluster*	Meningkatkan kemampuan cluster basis data dengan metode *horizontal scalling*	menggunakan spesifik database milik facebook (FQL)	Menggunakan database PostgreSQL yang diimplementasikan pada kontainer
2	Gibadullin R.F., Vershinin I.S., Minyazev R.Sh. [2017]. *Realization of Replication Mechanism in PostgreSQL DBMS*	Meneliti mekanisme replikasi sistem basis data PostgreSQL	Melakukan replikasi pada server virtual berbasis windows server	Melakukan replikasi pada virtualisasi berbasis kontainer
3	David Bernstein [2014], Containers and Cloud: From LXC to Docker to Kubernetes	Meneliti mengenai penggunaan docker kontainer didalam Kubernetes	Melakukan implementasi dari LXC kemudian menggunakan docker hingga Kubernetes	Melakukan implementasi kontainer dengan menggunakan docker yang dikelola oleh Kubernetes yang

				berisi aplikasi basis data PostgreSQL

2.2 Dasar teori

2.2.1 Kontainer

Kontainer merupakan solusi yang *reliable* dari problem dalam menjalankan sebuah aplikasi yang bekerja secara berpindah-pindah seperti dari lingkungan pengembang kedalam lingkungan *production*. Mudahnya bahwa kontainer menjadi wadah dan melakukan isolasi terhadap aplikasi dengan segala kebutuhannya. Kontainer memiliki ukuran yang sangat kecil yang menyebabkan kontainer banyak digunakan oleh pengembang saat ini. Sistem kontainer sudah dikembangkan sejak lebih dari 10 tahun oleh komunitas linux yang disebut LXC (Breinstein, 2014).

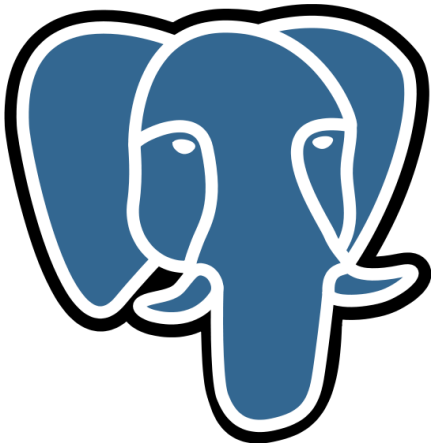
Kontainer memiliki kelebihan diantaranya :

1. Kontainer akan berbagi *resource* terhadap host OS dimana kontainer itu berdiri. Sehingga menjadikan kontainer memiliki waktu mulai yang cepat dan ringan.
2. Bersifat *portable* sehingga memudahkan pengembang dalam melakukan *deploying* didalam berbagai sistem operasi yang digunakan selama sistem operasi tersebut mendukung berjalannya kontainer.
3. Mempercepat waktu *deployment* dikarenakan seluruh kebutuhan terkait aplikasi yang akan di jalankan telah diisolasi didalam kontainer yang disebut *images*.

2.2.2 PostgreSQL

Didalam dunia basis data terdapat berbagai macam jenis basis data *server* berbasis *Structured Query Language (SQL)*. PostgreSQL merupakan sebuah sistem basis data *relational open source* yang telah aktif dikembangkan lebih dari 20 tahun. PostgreSQL memiliki kehandalan, integritas data, ketahanan fitur dan kemudahan yang dapat diperoleh oleh pengguna (Crunchy-data, 2017).

Perbedaan paling mendasar antara PostgreSQL dengan sistem basis data relasional lainnya adalah kemampuan PostgreSQL yang memungkinkan user untuk mendefinisikan SQL sendiri seperti membuat *function*. Hal ini dapat dilakukan karena PostgreSQL tidak hanya menggunakan tabel dan kolom namun juga terdapat tipe, fungsi dan informasi lainnya. Semua fitur tersebut dikumpulkan dalam sebuah *class* yang memungkinkan user dapat melakukan perubahan. Dengan model *class* seperti ini akan memudahkan PostgreSQL dapat dikembangkan oleh pengguna. Pada database lainnya diperlukan modul tambahan atau dengan mengganti kode program untuk dapat seperti yang PostgreSQL lakukan. Seperti pada gambar 2.1 PostgreSQL memiliki logo berupa gambar gajah yang disebut dengan *slonik* (PostgreSQL, n.d.).



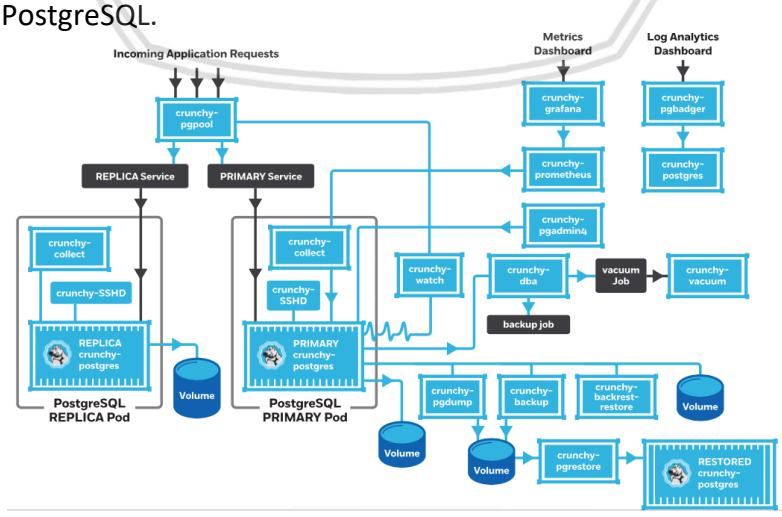
Gambar 2.1 Logo PostgreSQL (sumber: postgresql.org)

PostgreSQL memiliki kelemahan salah satunya dalam permasalahan replikasi. Replikasi belum disertakan dalam distribusi standart PostgreSQL (Fikriansyah, 2016). Namun pada pembahasan didalam penelitian ini akan mencoba melakukan implementasi replikasi postgreSQL untuk meningkatkan kemampuan load PostgreSQL sehingga mampu meningkatkan ketersediaan tingkat tinggi dengan metode *Load Balancing*.

2.2.3 Crunchy data

Crunchy Data merupakan pemimpin industri dalam bidang PostgreSQL secara *enterprise* dan juga merupakan solusi sumber terbua (Crunchy-data, 2017). Crunchy data didirikan pada tahun 2012 dengan membawa *mission-critical* untuk meningkatkan kemampuan dari sistem basis data PostgreSQL diantaranya dengan meningkatkan kemampuan *scalling* secara besar.

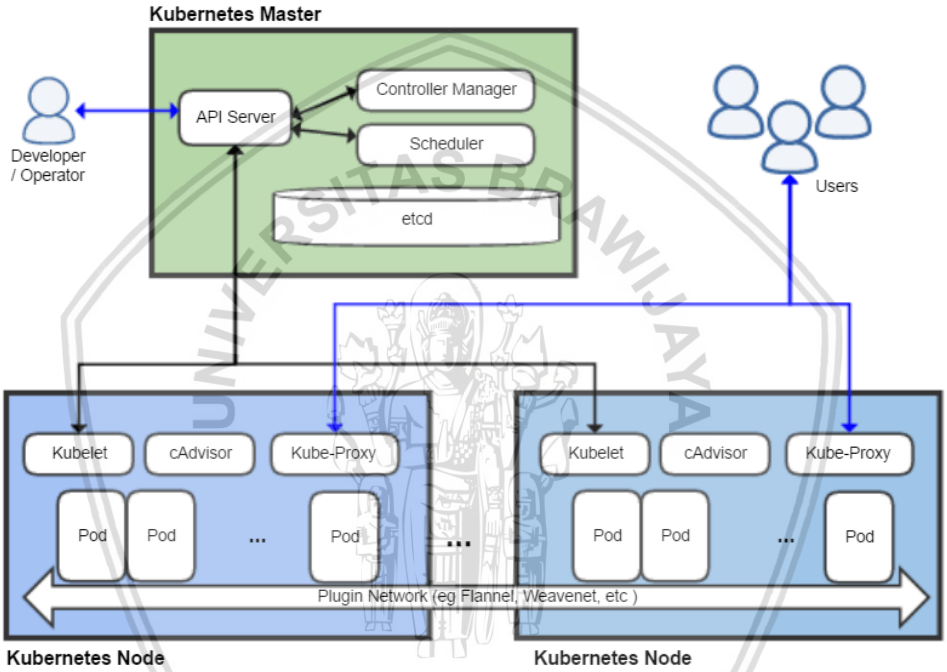
Crunchy Data berkomitmen untuk turut aktif mengembangkan basis data PostgreSQL dalam pengembangan secara *global*. Segala pengembangan yang dilakukan crunchy data merupakan murni untuk sumber terbuka. Pada gambar 2.2 menjelaskan bagaimana crunchy data menjalankan sebuah cluster sistem basis data milik PostgreSQL.



Gambar 2.2 PostgreSQL-*as-a-service* oleh crunchy data

2.2.4 Kubernetes

Kubernetes adalah sebuah sistem *open source* untuk mengoptimalisasi penyebaran, scalling dan pengelolaan *container*. pengelompokan *container* yang terbentuk dalam *logic unit* untuk mempermudah pengelolaan. Kubernetes sudah berdiri selama 15 tahun dan memiliki pengalaman dalam *production* di Google (kubernetes, 2018). Kubernetes dirancang dengan prinsip yang memungkinkan Google menjalankan miliaran kontainer dalam seminggu. Kubernetes dapat mempermudah tim ops tanpa harus memperbanyak personil dari tim ops. Pada gambar 2.3 digambarkan arsitektur Kubernetes berjalan dalam melayani permintaan pengguna. Pengguna akan berhubungan dengan *kube-proxy* untuk mendapatkan kebutuhan yang diinginkan dari aplikasi didalam pod.

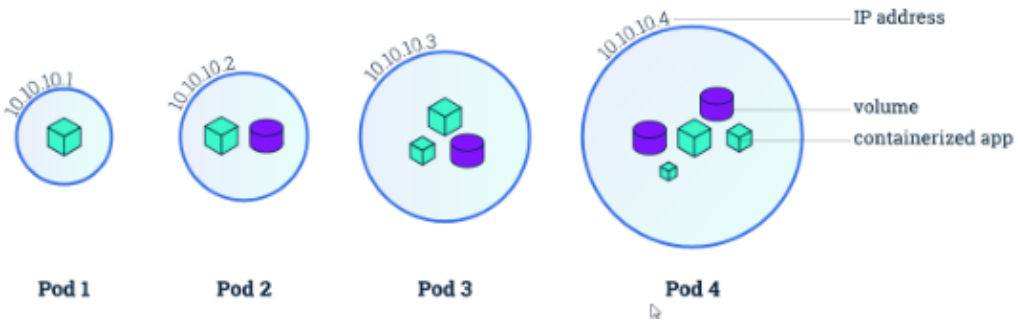


Gambar 2.3 Arsitektur Kubernetes (sumber: nxgcloud.com)

Ada beberapa istilah yang terdapat pada Kubernetes. diantaranya adalah

- POD

POD adalah satu grup container instance. Kita bisa menjalankan beberapa container dalam satu pod. Pada setiap pods memiliki IP yang digunakan sebagai alamat untuk berkomunikasi seperti yang digambarkan pada gambar 2.4. Container satu dengan satu pod yang sama dapat saling berkomunikasi dan saling mengakses. Pod bersifat sementara karena pod ini dapat dihapuskan dan dapat dibuat sesuai dengan kebutuhan yang diinginkan (Madiraju, 2018)



Gambar 2.4 Arsitektur Pod (sumber: nxgcloud.com)

- *Node*

Node adalah representasi dari suatu mesin server. Node dapat berupa virtual mesin ataupun mesin fisik.

- *Service*

Merupakan mekanisme untuk mengakses POD dari luar / publik. Aplikasi yang berjalan dalam POD tidak memiliki alamat *static IP*. Namun agar dapat diakses oleh user atau aplikasi maka diperlukan alamat *static IP*. *Service* menyediakan alamat *IP static* yang nantinya dapa diarahkan kedalam POD dengan menggunakan selector (Madiraju, 2018).

- Label

Label merupakan seperangkat informasi *metadata* yang digunakan untuk mencari sebuah POD tertentu

- *API Server*

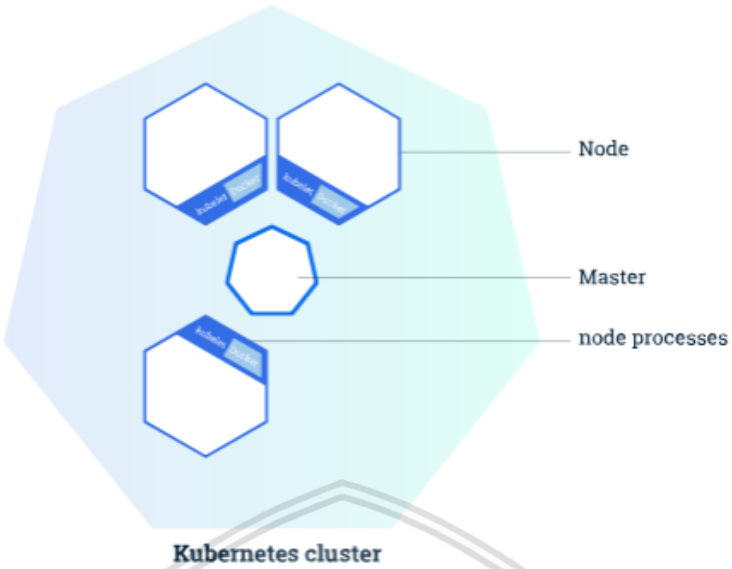
Api server berfungsi sebagai pintu masuk segala perintah yang digunakan untuk menjalankan segala aktifitas didalam Kubernetes.

- *Selector*

Selector merupakan sebuah cara untuk melakukan *filtering* dalam upaya mencari pod berdasarkan label.

2.2.5 Kubernetes cluster

Kubernetes *cluster* merupakan sekumpulan server yang saling terhubung terhadap suatu unit sistem atau lingkungan. Kubernetes melakukan *deployment container* secara otomatis kedalam cluster tanpa melihat perlu server satu persatu. Kubernetes *cluster* memiliki 2 jenis *resource.* Yaitu *master* sebagai *coordinator cluster* yang berfungsi sebagai *coordinator* sistem dan *worker* yang berkerja untuk menjalankan berbagai macam aplikasi didalam kontainer. Master dapat diletakan di server mana saja sehingga akan dapat ditemui sebuah server memiliki dua fungsi yang berbeda yaitu sebagai *master* dan sebagai *worker*.

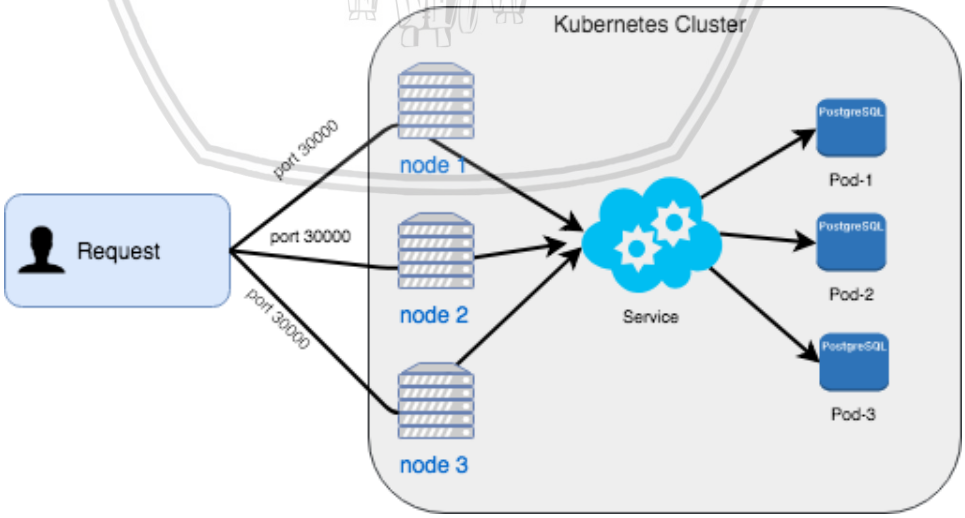


Gambar 2.5 Kubernetes cluster arsitektur (sumber: nxgcloud.com)

Pada gambar 2.5 Dalam setiap node memiliki sebuah sistem untuk menjalankan kontainer. Ada beberapa opsi yang dapat dipakai diantaranya adalah menggunakan docker.

2.2.6 Nodeport

Nodeport service merupakan sebuah sistem untuk meneruskan permintaan terhadap sebuah *service* kedalam *cluster* Kubernetes berdasarkan port. Pada gambar 2.6 menjelaskan bahwa pengguna mengakses server yang terhubung didalam Kubernetes *cluster*. Kemudian dari node tersebut akan diarahkan kedalam service Kubernetes. Selanjutnya service tersebutlah yang akan meneruskan permintaan pengguna kedalam pod pemrosesan.



Gambar 2.6 Nodeport proses

2.2.7 Load balancer

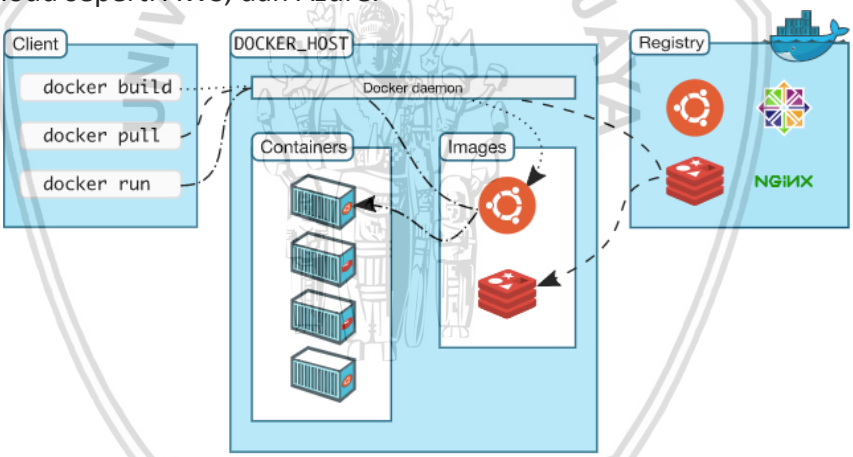
Kubernetes dapat melakukan load balancing terhadap request pada service di dalam lingkungan Kubernetes. Load balancer merupakan salah satu komponen

dari fungsi *load balancing* yang digunakan untuk mendistribusikan permintaan pengguna ke seluruh node maupun pod yang tersedia. *Load balancer* menggunakan algoritma *Round Robin* dalam menjalankan pekerjaanya untuk mendistribusikan permintaan pengguna (Simon, 2017).

2.2.8 Docker

Docker merupakan salah satu *open-platform software* dengan konsep kontainer. Docker terbuka untuk siapa saja yang ingin membangun, melakukan *deployment,* dan melakukan *management* berbagai macam aplikasi, terutama bagi para *developers, sysadmins,* atau *individual* di berbagai perangkat seperti laptop, *data center, virtual machine, atau cloud.* Dengan konsep kontainerisasi ini menjadikan docker cepat dan ringan untuk diimplementasikan. Docker berjalan di bawah lisensi Apache 2.0 dan memiliki komponen yang terdiri dari docker *client* dan *server*, *docker image*, *registry*, dan *docker container* (what is a container platform, n.d.).

Docker menerapkan sistem arsitektur aplikasi *client-server*, di mana *docker client* berkomunikasi dengan *docker server* (Docker daemon), kemudian *Docker daemon* yang bertanggungjawab untuk melakukan proses yang lainnya. Hingga saat ini, docker dapat dijalankan pada sistem operasi Linux, Mac, Windows, maupun cloud seperti AWS, dan Azure.



Gambar 2.7 Arsitektur Docker

Gambar 2.7 menggambarkan arsitektur docker dalam menjalankan kontainer. Berikut merupakan beberapa istilah dalam docker.

2.2.8.1 Docker daemon

Docker daemon memiliki fungsi untuk membangun, mendistribusikan dan menjalankan fungsi kontainer dari docker. User tidak dapat langsung menggunakan docker daemon, akan tetapi untuk menggunakan docker daemon maka user menggunakan docker client sebagai perantara atau Command Line Interface (CLI).

2.2.8.2 Docker images

Docker *images* merupakan sebuah *template* yang bersifat *read only*. Docker *image* berisi OS maupun aplikasi yang telah di install sebelumnya yang dijadikan dasar dari *instance* yang akan digunakan. Docker *images* berfungsi untuk

membuat docker container, dengan hanya 1 docker *images* kita dapat membuat banyak docker container.

2.2.8.3 Docker Container

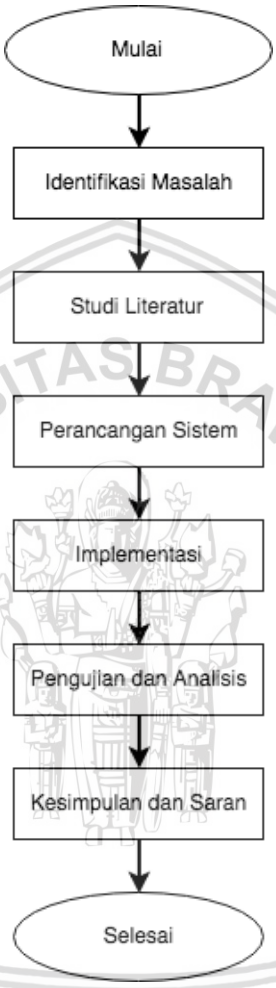
Docker *container* dapat dikatakan mirip dengan sebuah folder, dimana docker *container* ini dibuat dengan menggunakan docker *image*. Docker *container* berdiri diatas docker *image*. docker container inilah yang nantinya akan menjalankan aplikasi dan dilakukannya replika untuk menciptakan system cluster.

2.2.8.4 Docker Registry

Docker *registry* merupakan sebuah sistem yang berisi kumpulan dari docker *image* yang bersifat *private* maupun *public* yang dapat diakses di docker *hub*. Dengan menggunakan docker *registry*, tentunya memudahkan *sysadmin* maupun *developer* untuk menggunakan docker *image* yang telah dibuat oleh *developer* maupun *sysadmin* lain. Sehingga memudahkan serta mempercepat pekerjaan.

BAB 3 METODOLOGI

Metodologi penelitian menjelaskan bagaimana metodologi yang digunakan dalam melakukan penelitian. Dalam gambar 3.1 menjelaskan alur metodologi atau langkah yang digunakan penulis dalam melakukan penelitian.



Gambar 3.1 Metodologi Penelitian

3.1 Identifikasi masalah

Masalah yang diteliti dalam penelitian ini adalah bagaimana rancangan sistem *load balancing* server basis data pada virtualisasi berbasis kontainer yang mampu meningkatkan kinerja server basis data dalam memenuhi kebutuhan data yang diminta oleh pengguna. Dari permasalahan tersebut kemudian diindentifikasi cara-cara yang dapat digunakan untuk menyelesaikan permasalahan tersebut. Dari identifikasi permasalahan ini kemudian dijarbarkan solusi-solusi yang memungkinkan untuk diambil dan diterapkan. Setelah menentukan permasalahan yang diteliti kemudian dilanjutkan dengan mencari tujuan penelitian.

3.2 Studi literatur

Mempelajari literature dari berbadai bidang ilmu yang sesuai dengan penelitian. Diantaranya :

1. Kontainer Basis data PostgreSQL
2. Custering Kubernetes
3. *Pod Scalling*
4. *Network File System*
5. *Load balancing basis data*

Literatur tersebut didapat dari buku, jurnal, penelitian sebelumnya dan dokumentasi yang ditemukan selama pengerjaan skripsi.

3.3 Rekayasa kebutuhan dan perancangan

Menganalisis kebutuhan sistem untuk mempersiapkan dalam melakukan implementasi dari sebuah sistem yang sedang diteliti yaitu sistem *load balancing* server basis data pada virtualisasi berbasis kontainer. Sistem ini berjalan diatas 5 node server yang tergabung didalam Kubernetes cluster.

3.4 Implementasi sistem

Tahap implementasi merupakan tahap selanjutnya dari tahapan perancangan. Pada tahap ini sistem dibangun berdasarkan rancangan yang telah dibuat untuk membangun sistem *load balancing* server basis data pada virtualisasi berbasis kontainer.

3.5 Pengujian dan analisis

Pengujian sistem dilakukan setelah implementasi telah dilaksanakan. Tujuan dari uji coba sistem ini adalah memastikan hasil dari implementasi sistem yang telah dibuat sesuai dengan perancangan yang telah dilakukan. Apabila terjadi kesalahan yang menyebabkan kegagalan dalam ujicoba maka akan dilakukan kembali perancangan serta mengulang implementasi.

3.6 Kesimpulan

kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian metode telah diterapkan sudah dilakukan. Kesimpulan ini dari hasil pengujian dan analisis metode. Tahap akhir dari penuliasn ini adalah saran yang dimaksudkan untuk memperbaiki kesalahan yang terjadi serta memberikan pertimbangan untuk pengambangan selanjutnya

BAB 4 REKAYASA KEBUTUHAN DAN PERANCANGAN

Bab ini bertujuan untuk menjabarkan analisis kebutuhan dan perancangan dari sistem yang akan dibangun. Adapun kebutuhan yang dijabarkan adalah kebutuhan yang diperlukan untuk membangun sistem *load balancing* kontainer basis data PostgreSQL didalam Kubernetes.

4.1 Ruang lingkup

Sistem yang dibangun merupakan sebuah sistem basis data server yang berjalan didalam kontainer. Terdapat lima node server clustering yang dikelola oleh Kubernetes. Kemudian dilakukan *deploying* kontainer basis data PostgreSQL yang dilakukan menggunakan image milik Cruncydata. Kontainer basis data PostgreSQL di deploy didalam Kubernetes menggunakan *kind* berupa statefulset.

4.2 Analisis kebutuhan

Analis kebutuhan menjelaskan kebutuhan terkait dengan implementasi yang akan dilakukan. Analisis kebutuhan mencakup kebutuhan fungsional, kebutuhan perangkat keras dan kebutuhan perangkat lunak

4.2.1 Kebutuhan fungsional

Kebutuhan fungsional merupakan kebutuhan yang harus dipenuhi sistem agar sistem dapat berjalan dengan baik sesuai dengan tujuan penelitian. Berikut ini adalah beberapa penjelasan dari kebutuhan fungsional pada sistem ini.

1. Kubernetes *cluster* dapat berjalan dengan baik didalam 5 node.
2. PostgreSQL dapat dideploy didalam Kubernetes *cluster.*
3. Pod PostgreSQL dapat dilakukan replika sebanyak maksimal 5 pod.
4. Terdapat satu NFS server yang digunakan untuk menampung data.
5. Seluruh data didalam pod tidak terjadi perbedaan.
6. Pendistribusian permintaan oleh Kubernetes berjalan dengan baik.

4.2.2 Kebutuhan perangkat keras

Adapun kebutuhan perangkat keras adalah sebagai berikut :

1. Deployment Server dan NFS server
   - Type : Mesin Virtual
   - Processor : Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (4 Core)
   - Ram : 4GB Ram DDR4
   - Storage : 100GB 15K SAS
   - Alamat IP : 10.10.0.215
2. Node 1 (Master Node)
   - Type : Mesin Virtual
   - Processor : Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (4 Core)
   - Ram : 4GB Ram DDR4
   - Storage : 100GB 15K SAS

- Alamat IP : 10.10.0.216

3. Node 2 (Master Node, Worker Node)
   - Type : Mesin Virtual
   - Processor : Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (4 Core)
   - Ram : 4GB Ram DDR4
   - Storage : 100GB 15K SAS
   - Alamat IP : 10.10.0.217
4. Node 3 (Worker Node)
   - Type : Mesin Virtual
   - Processor : Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (4 Core)
   - Ram : 4GB Ram DDR4
   - Storage : 100GB 15K SAS
   - Alamat IP : 10.10.0.218
5. Node 4 (Worker Node)
   - Type : Mesin Virtual
   - Processor : Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (4 Core)
   - Ram : 4GB Ram DDR4
   - Storage : 100GB 15K SAS
   - Alamat IP : 10.10.0.219
6. Node 5 (Worker Node)
   - Type : Mesin Virtual
   - Processor : Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (4 Core)
   - Ram : 4GB Ram DDR4
   - Storage : 100GB 15K SAS
   - Alamat IP : 10.10.0.220

4.2.3 Kebutuhan perangkat lunak

Berikut merupakan perangkat lunak yang dibutuhkan untuk membangun sistem basis data PostgreSQL didalam Kubernetes yang mampu mempertahankan konsistensi data serta mampu melakukan *scalling* dan *load balancing* sehingga mencapai ketersediaan tingkat tinggi.

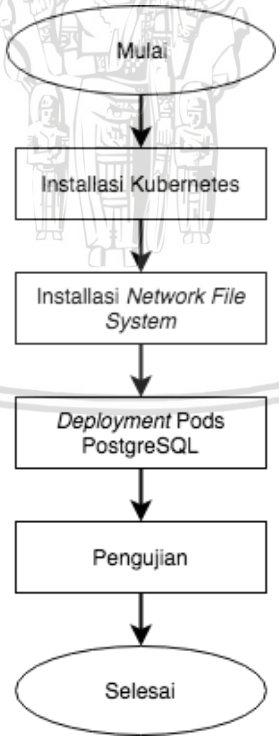
1. Deployment Server
   a. Linux Ubuntu 16.04
   b. NFS Server
   c. PostgreSQL client
2. Node 1
   a. Kubernetes 1.9
   b. NFS Client
   c. Docker 17.03
3. Node 2
   a. Kubernetes 1.9

b. NFS Client
c. Docker 17.03

4. Node 3
   a. Kubernetes 1.9
   b. NFS Client
   c. Docker 17.03
5. Node 4
   a. Kubernetes 1.9
   b. NFS Client
   c. Docker 17.03
6. Node 5
   a. Kubernetes 1.9
   b. NFS Client
   c. Docker 17.03

4.3 Perancangan sistem

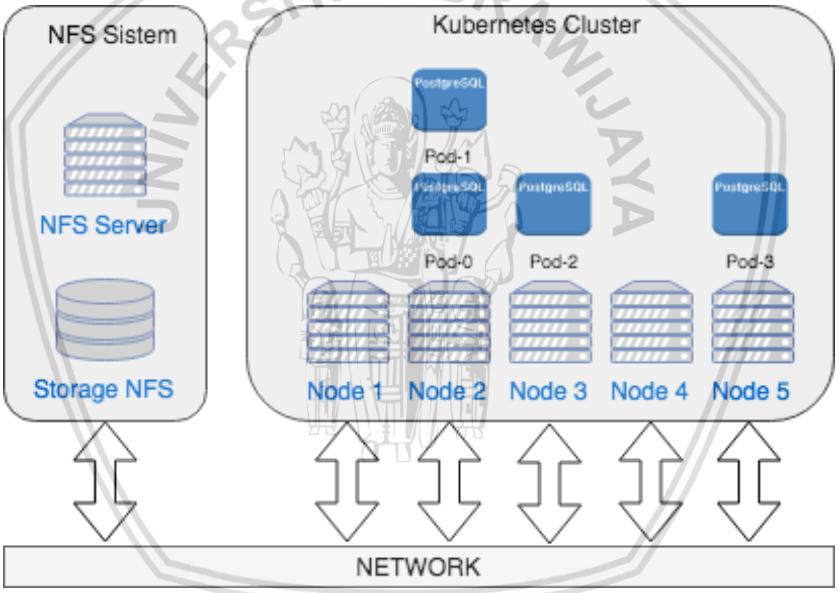
Dalam merancang sebuah sistem maka dibutuhkan sebuah tahapan yang akan dilakukan dalam melakukan implementasi sistem basis data PostgreSQL didalam Kubernetes yang mampu mempertahankan konsistensi data dan dapat melakukan *scalling* dan *loadbalancing* sehingga mencapai ketersediaan tingkat tinggi. Dibawah ini merupakan alur tahapan yang akan digunakan :



Gambar 4.1 Tahapan implementasi sistem

Pada gambar 4.1 dapat dilihat tahapan dalam melakukan implementasi sistem load balancing server basis data pada virtualisasi berbasis kontainer. Pembahasan umum dari masing-masing tahapan akan dijelaskan sebagai sebagai berikut :

1. Melakukan pemasangan Kubernetes didalam node yang telah disiapkan sebelumnya. Kubernetes dipasang menggunakan sistem kubespray untuk mempermudah.
2. Setelah sistem *cluster* Kubernetes telah berjalan baik ditandai dengan status *ready* oleh setiap node. Kemudian dilakukan pemasangan sebuah sistem sharing file yang disebut dengan *network file sistem(NFS)*. Network file sistem server dipasang didalam sebuah *server deployment* sebagai tempat untuk menampung segala data yang dilakukan oleh PostgreSQL nantinya. Kemudian dilakukan pemasangan NFS client di semua node disiapkan.
3. Setelah NFS berhasil dipasang, maka tahapan selanjutnya adalah melakukan *deployment* PostgreSQL server dengan menggunakan *image* yang dibuat oleh crunchy data.
4. Kemudian dilakukan pengujian terhadap cluster PostgreSQL *server* yang telah berhasil dimasukan kedalam sistem Kubernetes. pengujian yang dilakukan adalah memastikan sistem dapat berjalan dengan baik yaitu pengujian *scalling*, pengujian *loadbalancing,* dan pengujian konsistensi data.



Gambar 4.2 Rancangan arsitektur

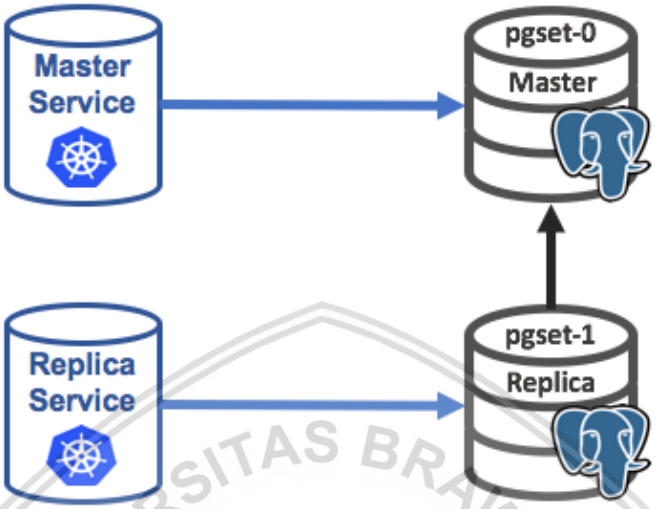
Gambar 4.2 menjelaskan secara keseluruhan dari arsitektur yang akan dibangun. Sesuai dengan urutan yang telah disebutkan bahwa nantinya sistem akan dibangun dengan arsitektur yang saling berhubungan antar komponen. Antar node dihubungkan oleh jaringan. Kemudian sebagai penyimpanan data dihubungkan dengan *network file system (NFS)* melalui jaringan yang sama.

4.4 Perancangan skenario pengujian

Terdapat 3 sistem yang akan dilakukan pengujian untuk memastikan bahwa sistem berjalan sesuai dengan *goal* yang telah ditentukan dalam penelitian. 3 sistem terbut adalah *scalling*, *loadbalancing* dan konsistensi data.

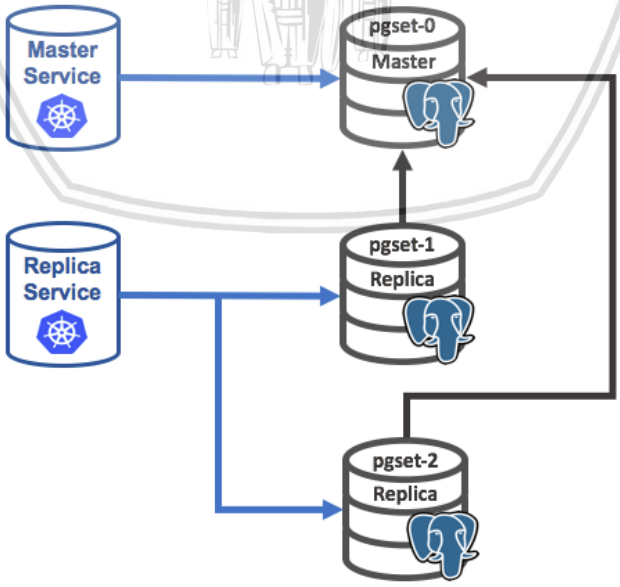
4.4.1 Skenario pengujian *scalling*

Scalling dilakukan dengan mereplika pod milik PostgreSQL. Replika dapat dikatakan berhasil apabila pod replika dapat berjalan dengan baik. Konsep replika dilakukan adalah seperti pada gambar diawah ini :



Gambar 4.3 Replika Pertama

Replikasi dilakukan dengan menggandakan pod yang telah ada ini merupakan replikasi awal. Seperti yang dilihat dalam gambar 4.2 adalah bukan *pod master* yang dimiliki oleh postgre melalui crunchy data. Namun merupakan pod replika milik PostgreSQL. Kemudian untuk melakukan replikasi selanjutnya cukup dengan menyalin pod replika milik PostgreSQL.

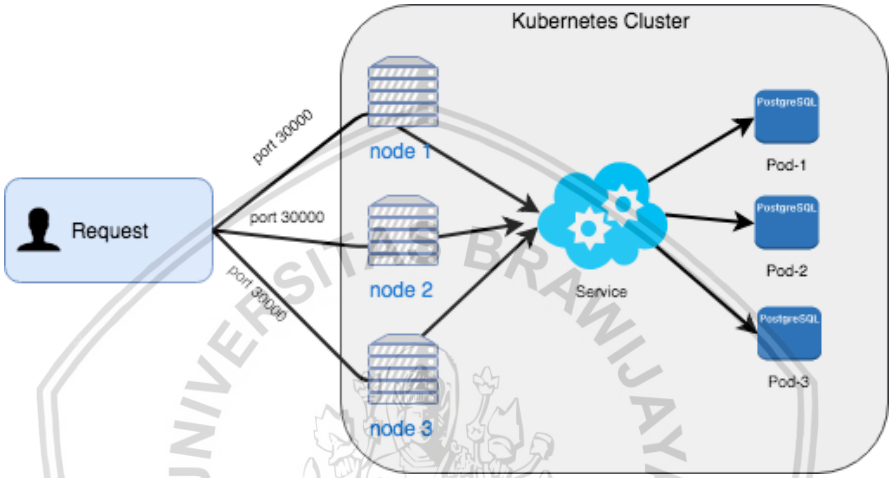


Gambar 4.4 Replikasi tahap selanjutnya

Pada gambar 4.3 memperjelas bahwa replikasi selanjutnya merupakan replikasi pada *replica-service* saja. Bukan melakukan replika pada *master service. Replicaset* melakukan streaming data pada basis data master.

4.4.2 Skenario pengujian *loadbalancing*

Loadbalancing dilakukan oleh Kubernetes. Dalam metode *loadbalancing* digunakan *NodePort* yang telah disediakan oleh Kubernetes. *Nodeport* menggunakan algoritma round robin dalam menjalankan *load balancing* (Simon, 2017).



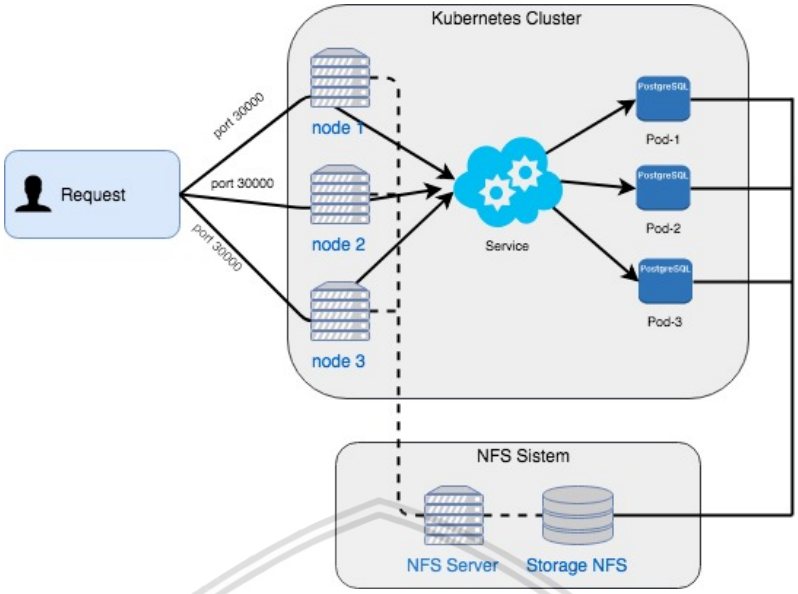
Gambar 4.5 Metode Nodeport

Pada gambar 4.4 sebuah request dapat diakses melalui semua *node* yang tergabung didalam *cluster* Kubernetes. kemudian request tersebut akan dimasukan kedalam service yang tersedia didalam Kubernetes. request akan didistribusikan oleh service kepada pod yang aktif.

Dalam pengujian *load balancing* akan dilakukan menggunakan *tool* yang bernama jmeter guna melakukan *generate* request secara besar. Sebelum dilakukan pengujian terlebih dahulu dipasang aplikasi wireshark untuk memastikan bahwa data yang diminta masuk kedalam kontainer yang tepat. Namun sebelum melakukan pengujian ini akan dibuat sebuah *dummy content database*

4.4.3 Skenario pengujian konsistensi data

Data yang disimpan oleh sistem *cluster* diharuskan konsisten. Karena sebuah sistem *cluster* akan memilih secara random oleh *loadbalancing* berdasarkan *load* yang sedang terjadi. Apabila data yang dihasilkan tidak konsisten maka akan menimbulkan permasalahan kedepannya.



Gambar 4.6 skenario penyimpanan data

Data disimpan didalam sebuah server storage. Pada penelitian ini digunakan *Network File System (NFS)* untuk menjaga konsistensi data pada pods yang aktif. Hal ini memungkinkan data diambil dari sumber penyimpanan yang sama. Dalam menguji konsistensi data akan diambil sample secara random dengan membandingkan output yang dihasilkan dari pengujian *load balancing.*

BAB 5 IMPLEMENTASI

Implementasi sistem akan dilaksanakan jika perancangan sudah dilakukan. Implementasi akan mengacu pada perancangan yang dibuat. Implementasi yang dilakukan akan menggunakan arsitektur yang sama dengan yang ada didalam perancangan.

5.1 Installasi Kubernetes

Kubernetes diinstall didalam lima *server node* yang tersedia. Dalam installasi Kubernetes digunakan sebuah tools yaitu kubespray. Versi yang Kubernetes yang digunakan adalah versi 1.9.5. Kubernetes diinstall di dalam lima *node* dengan pembagian tugas sebagai berikut

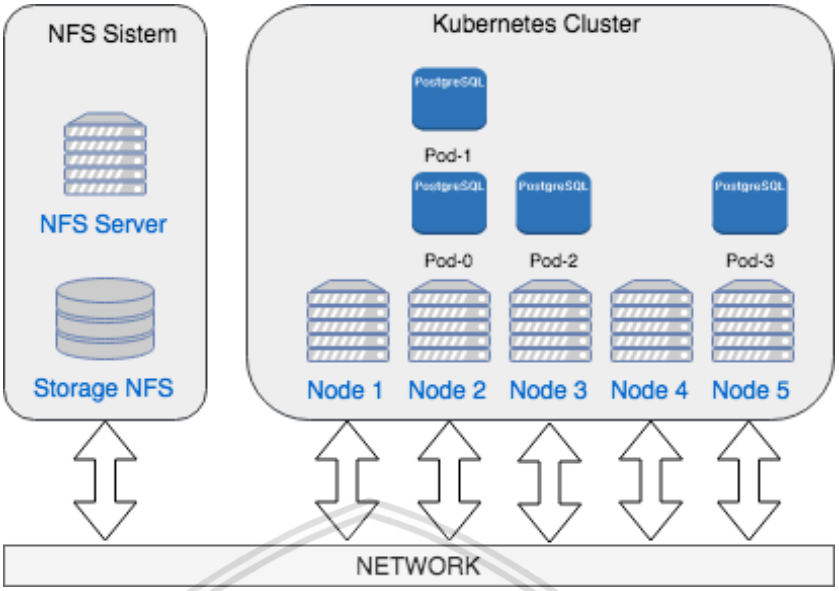
- Master Node
  Master node berfungsi sebagai koordinator yang mengatur segala pekerjaan yang dilakukan Kubernetes. Master node diinstall didalam node server 1 dan node server 2. Master node diinstall didalam 2 server karena untuk meningkatkan kemampuan Kubernetes dalam menjalankan pekerjaan.

- Worker Node
  Worker node berfungsi sebagai pekerja yang akan menjalankan berbagai kontainer yang akan dibuat oleh koordinator yaitu master node. Worker node akan menjalankan kontainer yang dijalankan didalam docker service yang dikelola oleh Kubernetes dalam bentuk pod. Worker inilah yang menjalankan pod dari PostgreSQL. Worker node diinstall didalam empat node server. Yaitu node server 2, node server 3, node server 4 dan node server 5.

- Etcd
  Etcd berfungsi sebagai tempat untuk menyimpan berbagai konfigurasi yang dilakukan oleh Kubernetes. Etcd diinstall didalam 3 node server. Yaitu node server 1, node server 2 dan node server 3.

5.2 Installasi *network file system*

Disiapkan sebuah server khusus untuk dijadikan server penyimpanan menggunakan *Network File System (NFS).* NFS diinstall didalam *server* diluar dari Kubernetes *cluster. Node server cluster* Kubernetes hanya sebagai sebagai client dari NFS. Sehingga segala transaksi data akan disimpan didalam NFS server. Pada gambar 5.1 menggambarkan alur penyimpanan data yang dilakukan oleh NFS.



Gambar 5.1 Network File System (NFS)

5.3 *Deploying* kontainer basis data PostgreSQL

Dalam *deploying* PostgreSQL untuk penelitian ini menggunakan *image* yang disediakan oleh crunchy data. berikut merupakan konfigurasi yml untuk melakukan *deploying* PostgreSQL.

Tabel 5.1 file konfigurasi service-account.yml

service-account.yml

```
1  apiVersion: v1
2  kind: ServiceAccount
3  metadata:
4    creationTimestamp: null
5    name: statefulset-sa
```

Konfigurasi pada tabel 5.1 merupakan konfigurasi untuk membuat *account* terhadap *service* yang akan dibuat. Dalam tabel 5.1 penulis membuat *account* bernama statefulset-sa.

Tabel 5.2 file konfigurasi net-service-primary.yml

net-service-primary.yml

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    creationTimestamp: null
5    labels:
6      name: statefulset-primary
7    name: statefulset-primary
8  spec:
9    externalIPs:
10   - 10.10.0.216
11   externalTrafficPolicy: Cluster
12   ports:
13   - nodePort: 31777
14     port: 5432
15     protocol: TCP
16     targetPort: 5432
17   selector:
```

```
18      name: statefulset-primary
19    sessionAffinity: None
20    type: LoadBalancer
21  status:
22    loadBalancer: {}
```

Kemudian konfigurasi pada tabel 5.2 berfungsi untuk membuat *service* didalam Kubernetes. *Service* yang akan dibuat merupakan *service* basis data primary dengan menggunakan port 31777 dan dengan label statefulset-primary.

Tabel 5.3 file konfigurasi net-servive-replica.yml

net-servive-replica.yml

```
1   apiVersion: v1
2   kind: Service
3   metadata:
4     creationTimestamp: null
5     labels:
6       name: statefulset-replica
7     name: statefulset-replica
8   spec:
9     externalIPs:
10    - 10.10.0.216
11    externalTrafficPolicy: Cluster
12    ports:
13    - nodePort: 31999
14      port: 5432
15      protocol: TCP
16      targetPort: 5432
17    selector:
18      name: statefulset-replica
19    sessionAffinity: None
20    type: LoadBalancer
21  status:
22    loadBalancer: {}
```

Kemudian konfigurasi pada tabel 5.3 berfungsi untuk membuat *service* didalam Kubernetes. *Service* yang akan dibuat merupakan *service* basis data *replica* dengan menggunakan port 31999 dan dengan label statefulset-replica. Nantinya service replica ini yang akan menjalankan *load balancing* menggunakan node port.

Tabel 5.4 file konfigurasi volume.yml

volume.yml

```
1   apiVersion: v1
2   kind: PersistentVolume
3   metadata:
4     creationTimestamp: null
5     labels:
6       name: statefulset-pgdata
7     name: statefulset-pgdata
8   spec:
9     accessModes:
10    - ReadWriteMany
11    capacity:
12      storage: 1Gi
13    nfs:
14      path: /nfsfileshare/skripsi-imam
15      server: 10.10.0.215
16    persistentVolumeReclaimPolicy: Retain
```

```
17  status:
18    phase: Pending
```

Volume.yml pada tabel 5.4 digunakan untuk menyiapkan *persistent volume* yang digunakan oleh kontainer basis data PostgreSQL sebesar 1 Gigabit. *Persistent volume* menggunakan sistem NFS yang terhubung pada NFS server di alamat ip 10.10.0.215 dengan *mount directory* yang terletak pada /nfsfilesharing/skripsi-imam.

Tabel 5.5 file konfigurasi volume-claim.yml

volume-claim.yml

```
1   apiVersion: v1
2   kind: PersistentVolumeClaim
3   metadata:
4     creationTimestamp: null
5     name: statefulset-pgdata
6   spec:
7     accessModes:
8     - "ReadWriteMany"
9     resources:
10      requests:
11        storage: 400M
12    selector:
13      matchLabels:
14        name: statefulset-pgdata
15  status:
16    phase: Pending
```

Volume-claim.yml pada tabel 5.5 merupakan sebuah konfigurasi yang digunakan untuk melakukan klaim terhadap *volume* yang telah disediakan pada konfigurasi volume.yml

Tabel 5.6 file konfigurasi statefulset.yml

statefulset.yml

```
1   apiVersion: apps/v1
2   kind: StatefulSet
3   metadata:
4     creationTimestamp: null
5     labels:
6       app: statefulset
7     name: statefulset
8   spec:
9     podManagementPolicy: OrderedReady
10    replicas: 2
11    revisionHistoryLimit: 10
12    selector:
13      matchLabels:
14        app: statefulset
15    serviceName: statefulset
16    template:
17      metadata:
18        creationTimestamp: null
19        labels:
20          app: statefulset
21      spec:
22        containers:
23        - env:
24          - name: PG_PRIMARY_USER
25            value: primaryuser
```

```
          - name: PGHOST
            value: /tmp
          - name: PG_MODE
            value: set
          - name: PG_PRIMARY_HOST
            value: statefulset-primary
          - name: PG_REPLICA_HOST
            value: statefulset-replica
          - name: PG_PRIMARY_PORT
            value: "5432"
          - name: PG_PRIMARY_PASSWORD
            value: password
          - name: PG_USER
            value: testuser
          - name: PG_PASSWORD
            value: password
          - name: PG_DATABASE
            value: userdb
          - name: PG_ROOT_PASSWORD
            value: password
          image: crunchydata/crunchy-postgres:centos7-10.4-1.8.3
          imagePullPolicy: IfNotPresent
          name: statefulset
          ports:
          - containerPort: 5432
            name: postgres
            protocol: TCP
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
          - mountPath: /pgdata
            name: pgdata
        dnsPolicy: ClusterFirst
        restartPolicy: Always
        schedulerName: default-scheduler
        securityContext: {}
        serviceAccount: statefulset-sa
        serviceAccountName: statefulset-sa
        terminationGracePeriodSeconds: 30
        volumes:
        - name: pgdata
          persistentVolumeClaim:
            claimName: statefulset-pgdata
    updateStrategy:
      type: OnDelete
status:
  replicas: 0
```

Statefulset.yml pada tabel 5.6 merupakan file konfigurasi inti dalam proses *deploying* kontainer basis data PostgreSQL. Dengan menggunakan *image* yang disediakan oleh crunchy data kontainer *dideploy* didalam Kubernetes *cluster*. Didalam statefulset.yml juga disertakan berbagai *environment* yang dibutuhkan seperti konfigurasi *user* dan *password*.

Agar dapat mempermudah *deployment* maka dibuatlah sebuah *file shell* dengan nama run.sh untuk menjalankan seluruh file konfigurasi yml. Tabel 5.8 berikut merupakan sebuah file shell run.sh :

Tabel 5.7 file konfigurasi run.sh

Run.sh	
1	#DELETE
2	kubectl delete -f statefulset.yml
3	kubectl delete clusterrolebindings statefulset-sa
4	kubectl delete -f service-account.yml -f volume.yml -f volume-claim.yml -f net-servive-primary.yml -f net-servive-replica.yml
	#CREATE
5	kubectl apply -f service-account.yml
6	kubectl apply -f net-servive-primary.yml -f net-servive-replica.yml
7	kubectl apply -f volume.yml -f volume-claim.yml
8	kubectl create clusterrolebinding statefulset-sa --
9	clusterrole=cluster-admin --user=admin --user=kubelet -- group=system:serviceaccounts --namespace=skripsi-imam
10	kubectl apply -f statefulset.yml

Kemudian untuk melakukan deploying dijalankan perintah didalam table 5.8

Tabel 5.8 Perintah untuk melakukan deployment

Runing command	
1	$ ./run.sh

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini akan dilakukan pengujian sebagai hasil dari implementasi yang telah dilakukan pada bab 5. Pengujian dilakukan berpedoman pada apa yang telah dirancang pada bab 4. Adapun hal yang akan dilakukan pengujian adalah *scalling*, *load balancing* dan konsistensi data.

6.1 Parameter pengujian

Dalam melakukan pengujian pada penelitian ini telah ditentukan parameter yang akan digunakan dalam melakukan pengujian. Parameter pengujian yang digunakan adalah sebagai berikut :

1. Pengujian *Scalling* : pengujian fungsionalis sistem dengan melakukan percobaan replikasi maksimal 5 replika
2. Pengujian *Load Balancing* : pengujian kinerja *load balancing* dengan melakukan permintaan data dalam jumlah besar menggunakan tool Jmeter dan wireshark.
3. Pengujian konsistensi data : pengujian validasi sistem dengan mengambil sample data yang dihasilkan dalam pengujian *load balancing* untuk kemudian dibandingkan.

6.2 Pengujian *scalling*

6.2.1 Tujuan

Pengujian *scalling* dilakukan untuk memastikan bahwa kontainer PostgreSQL didalam Kubernetes yang berbentuk pod dapat direplika dan berjalan dengan baik. Maksimal percobaan replikasi adalah sebanyak 5 replika.

6.2.2 Prosedur pengujian

Prosedur yang dilakukan untuk melakukan pengujian scalling adalah dengan menjalankan perintah berdasarkan kebutuhan replika. Untuk dapat melakukan replikasi sesuai dengan kebutuhan dapat dijalankan dengan perintah

Tabel 6.1 Perintah untuk melakukan replikasi

${CCP_CLI} scale --replicas=5 statefulset statefulset

Keterangan :
--replicas : merupakan parameter jumlah total replika yang diinginkan.

Pada setiap pengujian akan menggunakan perintah pada tabel 6.1 dengan melakukan perubahan pada parameter *–-replicas* secara bertahap. Yaitu melakukan replikasi sebanyak tiga replika, kemudian empat replika dan yang terakhir lima replika.

6.2.3 Hasil dan analisis

Hasil dari pengujian scalling terhadap pod basis data PostgreSQL dapat dijabarkan sebagai berikut

- Replikasi sebanyak 3 replika

```
Every 2.0s: kubectl get pods -o wide

NAME            READY     STATUS              RESTARTS   AGE       IP              NODE
pgadmin4-http   1/1       Running             0          1d        10.233.71.21    node3
pgbadger        2/2       Running             0          1d        10.233.74.86    node4
statefulset-0   1/1       Running             0          1d        10.233.74.87    node4
statefulset-1   1/1       Running             0          1d        10.233.71.23    node3
statefulset-2   0/1       ContainerCreating   0          1s        <none>          node5
```

Gambar 6.1 Proses replikasi 3 replika sedang berlangsung

Setelah perintah pada tabel 6.1 telah dilakukan maka Kubernetes cluster akan melakukan replikasi sebanyak 3 replika. Pada gambar 6.1 menunjukan proses replikasi sedang berlangsung yaitu dengan membuat kontainer baru yang diletakan pada node 2.

```
Every 2.0s: kubectl get pods -o wide

NAME            READY     STATUS    RESTARTS   AGE       IP              NODE
pgadmin4-http   1/1       Running   0          1d        10.233.71.21    node3
pgbadger        2/2       Running   0          1d        10.233.74.86    node4
statefulset-0   1/1       Running   0          1d        10.233.74.87    node4
statefulset-1   1/1       Running   0          1d        10.233.71.23    node3
statefulset-2   1/1       Running   0          3s        10.233.97.168   node5
```

Gambar 6.2 Proses replikasi 3 replika berhasil dilakukan

Kemudian pada gambar 6.2 menunjukan bahwa kontainer telah berhasil dibuat dan kemudian dapat dijalankan dengan baik.

- Replikasi sebanyak 4 replika

```
Every 2.0s: kubectl get pods -o wide

NAME            READY     STATUS              RESTARTS   AGE       IP              NODE
pgadmin4-http   1/1       Running             0          1d        10.233.71.21    node3
pgbadger        2/2       Running             0          1d        10.233.74.86    node4
statefulset-0   1/1       Running             0          1d        10.233.74.87    node4
statefulset-1   1/1       Running             0          1d        10.233.71.23    node3
statefulset-2   1/1       Running             0          10s       10.233.97.168   node5
statefulset-3   0/1       ContainerCreating   0          0s        <none>          node2
```

Gambar 6.3 Proses replikasi 4 replika sedang berlangsung

Setelah perintah pada tabel 6.1 telah dilakukan maka Kubernetes cluster akan melakukan replikasi sebanyak 4 replika. Pada gambar 6.3 menunjukan proses replikasi sedang berlangsung yaitu dengan membuat kontainer baru yang diletakan pada node 5.

```
Every 2.0s: kubectl get pods -o wide

NAME            READY     STATUS    RESTARTS   AGE       IP              NODE
pgadmin4-http   1/1       Running   0          1d        10.233.71.21    node3
pgbadger        2/2       Running   0          1d        10.233.74.86    node4
statefulset-0   1/1       Running   0          1d        10.233.74.87    node4
statefulset-1   1/1       Running   0          1d        10.233.71.23    node3
statefulset-2   1/1       Running   0          16s       10.233.97.168   node5
statefulset-3   1/1       Running   0          6s        10.233.75.13    node2
```

Gambar 6.4 Proses replikasi 4 replika berhasil dilakukan

Kemudian pada gambar 6.4 menunjukan bahwa kontainer statefulset-3 telah berhasil dibuat dan kemudian dapat dijalankan dengan baik.

- Replikasi sebanyak 5 replika

```
Every 2.0s: kubectl get pods -o wide

NAME            READY     STATUS              RESTARTS   AGE       IP              NODE
pgadmin4-http   1/1       Running             0          1d        10.233.71.21    node3
pgbadger        2/2       Running             0          1d        10.233.74.86    node4
statefulset-0   1/1       Running             0          1d        10.233.74.87    node4
statefulset-1   1/1       Running             0          1d        10.233.71.23    node3
statefulset-2   1/1       Running             0          23s       10.233.97.168   node5
statefulset-3   1/1       Running             0          13s       10.233.75.13    node2
statefulset-4   0/1       ContainerCreating   0          2s        <none>          node4
```

Gambar 6.5 Proses replikasi 5 replika sedang berlangsung

Setelah perintah pada tabel 6.1 telah dilakukan maka Kubernetes cluster akan melakukan replikasi sebanyak 5 replika. Pada gambar 6.5 menunjukan proses replikasi sedang berlangsung yaitu dengan membuat kontainer baru yang diletakan pada node 4.

```
Every 2.0s: kubectl get pods -o wide

NAME            READY     STATUS    RESTARTS   AGE       IP              NODE
pgadmin4-http   1/1       Running   0          1d        10.233.71.21    node3
pgbadger        2/2       Running   0          1d        10.233.74.86    node4
statefulset-0   1/1       Running   0          1d        10.233.74.87    node4
statefulset-1   1/1       Running   0          1d        10.233.71.23    node3
statefulset-2   1/1       Running   0          27s       10.233.97.168   node5
statefulset-3   1/1       Running   0          17s       10.233.75.13    node2
statefulset-4   1/1       Running   0          6s        10.233.74.98    node4
```

Gambar 6.6 Proses replikasi 5 replika berhasil dilakukan

Kemudian pada gambar 6.6 menunjukan bahwa kontainer telah berhasil dibuat dan kemudian dapat dijalankan dengan baik.

6.3 Pengujian *load balancing*

6.3.1 Tujuan

Pengujian load balancing dilakukan untuk memastikan bahwa load balancing kontainer PostgreSQL didalam Kubernetes berjalan dengan baik dengan dapat mendistribusikan permintaan data kepada seluruh pod sehingga meningkatkan kinerja server basis data.

6.3.2 Prosedur pengujian

Prosedur pengujian yang dilakukan adalah dengan melakukan pengujian beban sebanyak 4 kali pada setiap replika. Dari setiap pengujian dijalankan sebuah *query select* seperti pada tabel 6.2.

Tabel 6.2 Query Uji beban

select * from authors limit 1000;

Parameter yang akan diujikan adalah :

- Beban diujikan secara bertahap sebesar 100, 250, 500, 750 *Request per second*

- Mengambil data dengan limit 1000 Baris dari 147609 yang tersedia

Pengujian dilakukan menggunakan tool Jmeter dengan mengambil hasil dari *Troughput* Dengan *satuan request per second (TPS), Receiving speed* dengan satuan *kb per second* (kb/s), persentase *error* dengan satuan *percentage (%) dan* distribusi beban dengan satuan *(requests).* Kemudian dengan menggunakan *tools* wireshark dapat kita lihat distribusi data terhadap permintaan kepada pod yang tersedia.

6.3.3 Hasil dan analisis

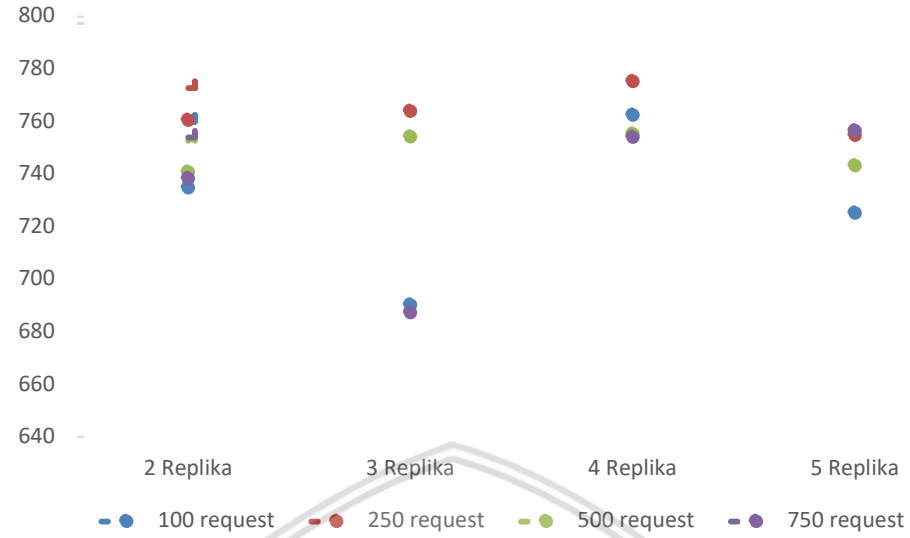
Berdasarkan hasil pengujian didapatkan hasil sebagai berikut

1. *Troughput*

Gambar 6.7 Hasil *troughput* pengujian *load balancing*

Berdasarkan pada gambar 6.7 didapatkan hasil pengujian troughput dalam satuan *request per second.* Semakin besar request akan menghasilkan *troughput* yang semakin besar pula disebabkan banyaknya data yang diminta. Selain itu banyaknya replikasi juga dapat mempengaruhi hasil *troughput* sistem server basis data dalam melayani permintaan data sehingga semakin banyak unit pemrosesan yang mampu melayani permintaan data maka akan menghasilkan *troughput* yang semakin besar. Terjadi kenaikan ketika dilakukan penambahan replika. Namun terjadi anomaly pada replika ke-3 yang disebabkan faktor eksternal seperti sitem *cache* yang dimiliki oleh aplikasi basis data PostgreSQL (kerstiens, 2012) juga sistem *index* yang dilakukan oleh aplikasi basis data PostgreSQL (Winand, n.d.).

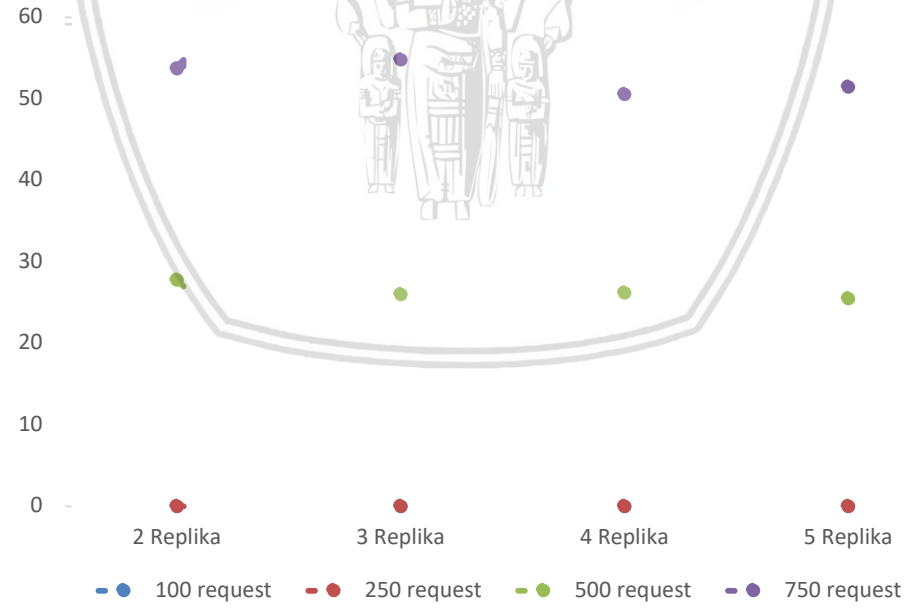
2. *Receiving Speed*



Gambar 6.8 Hasil *receiving speed* pengujian *load balancing*

Berdasarkan gambar 6.8 terjadi beberapa anomali. Hal ini dipengaruhi oleh koneksi yang dialami penguji dikarenakan dalam pengujian ini menggunakan koneksi *virtual private network*.

3. Persentase Error



Gambar 6.9 Hasil presentase *error* pengujian *load balancing*

Berdasarkan gambar 6.9 menunjukan penurunan persentase error yang terjadi didalam penerimaan data dari permintaan yang diberikan. Hal ini menunjukan bahwa kemampuan server basis data yang digunakan adalah berkisar pada 250 *request* per detik. Dikarenakan ketika dilakukan pengujian dengan beban

500 request per detik mulai menunjukan presentase *error*. Hal ini dikarenakan keterbatasan kemampuan server yang digunakan didalam penelitian ini.

4. Distribusi beban

Pada pengujian distribusi beban didapatkan hasil distribusi permintaan yang dilakukan user kepada sistem. Pengujian distribusi beban dilakukan mengambil pada pengujian 100 *requests* dan 250 *requests* karena pada pengujian tersebut tidak memiliki persentase *error*.

Tabel 6.3 Distribusi beban pada pengujian 100 *requests*

100 request	Replika 1	Replika 2	Replika 3	Replika 4	Replika 5
2 replika	48 request	52 request	-	-	-
3 replika	35 request	34 request	31 request	-	-
4 replika	25 request	26 request	24 request	25 request	-
5 replika	21 request	22 request	19 request	21 request	17 request

Pada tabel 6.3 menunjukan distribusi permintaan yang diminta oleh pengguna. Distribusi permintaan tersebut dilakukan oleh *service* Kubernetes dengan *kind load balancer*. Distribusi terbagi secara rata kepada seluruh replikasi. Namun sedikit terjadi anomali pada replika ke 3 dimana mendapat permintaan lebih kecil dibandingkan yang lain. Hal ini seperti pada pengujian troughput yaitu memiliki pengaruh eksternal seperti *cache* dan *index*.

Tabel 6.4 Distribusi beban pada pengujian 250 *requests*

250 request	Replika 1	Replika 2	Replika 3	Replika 4	Replika 5
2 replika	126 request	124 request	-	-	-
3 replika	84 request	82 request	83 request	-	-
4 replika	63 request	63 request	62 request	62 request	-
5 replika	50 request	51 request	48 request	51 request	50 request

Pada tabel 6.4 menunjukan distribusi permintaan yang diminta oleh pengguna. Distribusi permintaan pada pengujian 250 request tersebut dilakukan oleh *service* Kubernetes dengan *kind load balancer.* Distribusi terbagi secara rata kepada seluruh replikasi. Namun sedikit terjadi anomali pada replika ke 3 dimana mendapat permintaan lebih kecil dibandingkan yang lain. Hal ini seperti pada pengujian troughput yaitu memiliki pengaruh eksternal seperti *cache* dan *index*.

6.4 Pengujian konsistensi data

6.4.1 Pengujian

Pengujian konsistensi data dilakukan untuk memvalidasi kesesuaian data yang diterima dari sistem server basis data yang telah dirancang.

6.4.2 Prosedur pengujian

Prosedur pengujian yang dilakukan adalah dengan cara mengambil sample random dari beban pengujian 250 *request per second* dan 500 *request per second* yang diujikan sebanyak 10 data. Data tersebut diambil nilai besaran berdasarkan *Body size in byte* pada output yang dihasilkan oleh tool Jmeter

6.4.3 Hasil dan analisis

Berikut hasil dari pengujian validasi konsistensi data:

- Beban 250 request per detik

Tabel 6.5 Hasil validasi data pengujian *load balancing* pada beban 250 request

Data	*Body size in byte*			
	2 replika	3 replika	4 replika	5 replika
1	24308	24308	24308	24308
2	24308	24308	24308	24308
3	24308	24308	24308	24308
4	24308	24308	24308	24308
5	24308	24308	24308	24308
6	24308	24308	24308	24308
7	24308	24308	24308	24308
8	24308	24308	24308	24308
9	24308	24308	24308	24308
10	24308	24308	24308	24308

Berlandaskan hasil penelitian pada gambar 6.9 dan juga sejalan dengan hasil penelitian pada tabel 6.5 menunjukan bahwa pada beban uji sebesar 250 request per detik tidak menimbulkan error. Dan hasil dari besaran *byte* data yang diterima menunjukan data yang diterima merupakan data yang sama. Sehingga dapat dikatakan bahwa pada pengujian dengan beban 250 request per detik memiliki data yang valid pada setiap replikasi yang dilakukan.

- Beban 500 request per detik

Tabel 6.6 Hasil validasi data pengujian load balancing pada beban 500 request

Data	*Body size in byte*			
	2 replika	3 replika	4 replika	5 replika
1	24308	24308	24308	24308
2	0	24308	24308	24308
3	0	0	24308	0
4	24308	0	0	0
5	24308	24308	0	24308

6	24308	0	24308	24308
7	24308	24308	24308	24308
8	24308	24308	24308	24308
9	24308	24308	24308	24308
10	24308	24308	24308	24308

Berlandaskan hasil penelitian pada gambar 6.9 dan juga sejalan dengan hasil penelitian pada tabel 6.6 menunjukan bahwa pada beban uji sebesar 500 request per detik menghasilkan error. Dan hasil dari besaran *byte* data yang diterima menunjukan bahwa data yang diterima tidak seluruhnya diterima dengan baik. Hal ini disebabkan karena ketidakmampuan server basis data dalam memberikan data. Dengan melihat hasil yang didapatkan bahwa data yang diterima sebesar 0 *byte* tersebut menyatakan bahwa data tidak diterima dengan informasi *request time out*. Namun bukan menunjukan bahwa data yang diterima merupakan data yang tidak sama. Sehingga dapat dikatakan bahwa pada pengujian dengan beban 500 *request* per detik memiliki data yang valid pada setiap replikasi yang dilakukan.

BAB 7 PENUTUP

Pada bagian penutup ini dibahas mengenai kesimpulan dan saran terhadap penelitian yang telah dilakukan. Oleh penulis kesimpulan dan saran dijadikan terpisah, dengan penjelasan sebagai berikut:

7.1 Kesimpulan

Berdasarkan penelitian dengan judul implementasi *load balancing* basis data server pada virtualisasi berbasis kontainer dengan dilakukan beberapa pengujian didapatkan kesimpulan sebagai berikut :

1. Dapat merancangkan serta melakukan Implementasi sistem cluster kontainer basis data server pada virtualisasi berbasis kontainer telah berhasil dilakukan dan berjalan dengan baik berdasarkan pengujian fungsionalitas replikasi.
2. Sistem *load balancing cluster* basis data server pada virtualisasi berbasis kontainer berjalan dengan baik Sesuai dengan hipotesa yang telah dijelaskan didalam latar belakang bahwa dengan menambah unit pemprosesan seperti *instance, node* atau kontainer dalam bentuk pod dapat meningkatkan kemampuan server basis data dalam melayani lonjakan permintaan data dengan dibuktikan pada pengujian *load balancing* dan pengujian konsistensi data.
3. Tidak menutup kemungkinan akan terjadi *error* dalam melayani permintaan data. Namun dengan menambah unit pemrosesan dapat meningkatkan kinerja dari server basis data seperti yang telah dijabarkan pada bab pengujian. Berdasarkan pengujian yang telah dilakukan didapatkan hasil *troughput* pada pengujian *load balancing* dengan beban 750 request didapatkan hasil (61,3 request per detik) pada 2 replika, (63,1 request per detik) pada 3 replika, (64,6 request per detik) pada 4 replika, dan (65,6 request perdetik) pada 5 replika. Berdasarkan hasil pengujian distribusi data pada *load balancing* didapakan hasil distribusi data yang merata.

7.2 Saran

Dalam penelitian ini masih memiliki berbagai kekurangan. Dengan harapan pada penelitian selanjutnya dapat melakukan penyempurnaan terhadap penelitian ini. Adapun kekurangan tersebut antara lain adalah :

1. Dapat dilakukan implementasi dengan aplikasi server basis data yang lain sehingga dapat diketahui kinerja dari berbagai aplikasi server basis data.
2. Menggunakan penyimpanan terpusat lainnya seperti *glusterFS, infinit* maupun yang lainnya untuk meningkatkan kemampuan server basis data dalam memberikan data kepada pengguna.
3. Melakukan replikasi lebih banyak lagi untuk melihat kemampuan maksimal dari load balancing basis data PostgreSQL didalam virtualisasi berbasis kontainer

DAFTAR PUSTAKA

Affan, I. (2018, March 20). *Perkembangan Infrastruktur Saat ini*. Retrieved from Mico Citra Utama: https://cv-mcu.co.id/2018/03/20/perkembangan-infra-saat-ini.html

Breinstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE CLOUD COMPUTING*, 81-84.

Constantinov, C., Potreas, C. M., & Mocanu, M. L. (2016). Performing real-time social recommendations on a highly-available graph database cluster. *journal of IEEE*, 116-121.

Crunchy-data. (2017). *About Crunchy Data*. Retrieved July 2018, from Crunhy Data: https://www.crunchydata.com/about/

Dirgantara, A. (2016, September 10). *Teknologi Kontainer, Pengantar Pengenalan Docker*. Retrieved from Medium: https://blog.andi.dirgantara.co/teknologi-kontainer-pengantar-pengenalan-docker-706eafe03269

eMarketer. (2018). *Pengguna Internet Indonesia Nomor Enam Dunia*. Retrieved from Kementrian Komunikasi dan Informatika: https://kominfo.go.id/content/detail/4286/pengguna-internet-indonesia-nomor-enam-dunia/0/sorotan_media

Fikriansyah. (2016, December 23). *Tutorial Pedia*. Retrieved June 2018, from Apa itu PostgreSQL dan Sejarah Perkembangannya: https://www.tutorialpedia.net/apa-itu-postgresql-dan-sejarah-perkembangannya/

Gopinath P P, G., & Vasudevan, S. K. (2015). An in-depth analysis and study of Load balancing techniques in the cloud computing environment.

kerstiens, c. (2012, 10 1). *Understanding Postgres Performance*. Retrieved June 2018, from Craig Kerstiens: http://www.craigkerstiens.com/2012/10/01/understanding-postgres-performance/

Kubernetes. (2018). *Kubernetes*. Retrieved from Production-Grade Container Orchestration: https://Kubernetes.io/

Madiraju, N. (2018, June 19). *Kubernetes Architecture & Components*. Retrieved July 2018, from nxgcloud: http://nxgcloud.com/index.php/2018/06/19/Kubernetes-architecture-components/

PostgreSQL. (n.d.). *POSTGRESQL*. Retrieved July 2018, from the world's most advanced open source relational database: https://www.postgresql.org/

Prasetyo, B. (2018, 02 22). *Pendaftaran SNMPTN diwarnai server down*. Retrieved from edunews merawat kebhinekaan: https://www.edunews.id/edunews/kampus/pendaftaran-snmptn-diwarnai-server

Simon. (2017, March 15). *Kubernetes NodePort routing logic*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/38844215/Kubernetes-nodeport-routing-logic

Vasudevan, G. P. (2015). An In-depth Analysis and Study of Load Balancing Techniques in the Cloud Computing Environment. *2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)*, 427-432.

what is a container platform. (n.d.). Retrieved july 2018, from docker: https://www.docker.com/what-docker

Winand, M. (n.d.). *PostgreSQL Example Scripts for "Testing and Scalability"*. Retrieved june 2018, from use the index, luxe: https://use-the-index-luke.com/sql/example-schema/postgresql/performance-testing-scalability